John von Neumann Institute for Computing

Multigrid Smoothers on Multicore Architectures

Carlos García, Manuel Prieto, Fransisco Tirado

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr,
F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 279-286, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

Multigrid Smoothers on Multicore Architectures

Carlos García, Manuel Prieto, and Fransisco Tirado

Dpto. de Arquitectura de Computadores y Automática
Universidad Complutense
28040 Madrid, Spain
E-mail: {garsanca, mpmatias, ptirado}@dacya.ucm.es

We have addressed in this paper the implementation of red-black multigrid smoothers on high-end microprocessors. Most of the previous work about this topic has been focused on cache memory issues due to its tremendous impact on performance. In this paper, we have extended these studies taking *Multicore processors (MCP)* into account. With the introduction of *MCP*, new possibilities arise, which makes a revision of the different alternatives highly advisable. A new strategy is proposed which tries to achieve a cooperation between the threads in a *MCP*. Performance results on an *Intel Core*^{TM}*2 Duo* based system reveal that our alternative scheme can compete with and even improve sophisticated schemes based on loop fusion and tiling transformations aimed at improving temporal locality.

1 Introduction

Multigrid methods are regarded as being the *fastest* iterative methods for the solution of the linear systems associated with elliptic partial differential equations, and as amongst the *fastest* methods for other types of integral and partial differential equations[1]. *Fastest* refers to the ability of Multigrid methods to attain the solution in a computational work which is a small multiple of the operation counts associated with discretizing the system. Such efficiency is known as *textbook multigrid efficiency* (TME)[2] and has made multigrid one of the most popular solvers on the niche of large-scale problems, where performance is critical.

Nowadays, however, the number of executed operations is only one of the factors that influences the actual performance of a given method. With the advent of parallel computers and superscalar microprocessors, other factors such as *inherent parallelism* or *data locality* (i.e. the memory access behaviour of the algorithm) have also become relevant. In fact, recent evolution of hardware has exacerbated this trend since:

- The disparity between processor and memory speeds continues to grow despite the integration of large caches.
- Parallelism is becoming the key of performance even on high-end microprocessors, where multiple cores and multiple threads per core are becoming mainstream due to clock frequency and power limitations.

In the multigrid context, these trends have prompted the development of specialized multigrid-like methods[3–6], and the adoption of new schemes that try to bridge the processor/memory gap by improving locality[7–10]. Our focus in this paper is the extension of this cache-aware schemes to a *MCP*.

As its name suggests, *MCP* architectures integrate two or more processors (cores) on a chip. Its main goal is to enhance performance and reduce power consumption, allowing the

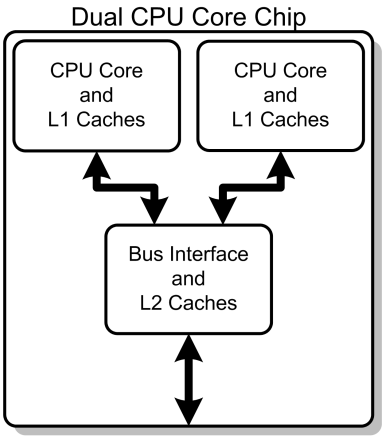

Figure 1. Intel Core^{TM}2 Duo block diagram.

simultaneous processing of multiple tasks in parallel. Technology trends indicate that the number of cores/processors on a chip will continue to grow. AMD has recently announced chips with four cores (*Native Quad technology*) and Intel has began to incorporate the Intel Core^{TM} Extreme quad-core Processor in their servers systems.

These cores can be seen as independent processors. Therefore, one may think that the optimizations targeted for *Symmetric Multiprocessors* (*SMP*) systems are also good candidates for *MCP* architectures. However, as shown in Fig. 1, most MCP architectures integrate a shared cache memory layer, which allows for a faster connection between on-chip cores. This sharing introduces an additional interaction between threads, which may translate into positive (fine-grain sharing among threads) or negative (competition for cache lines) effects. Conventional parallel schemes used in our context for SMP systems, such as block-based data distributions, may promote negative interactions and do not benefit from positive ones. Therefore, optimizations that are appropriate for these conventional machines may be inappropriate or less effective for *MCP*.

Unfortunately, *MCP* potentials are not yet fully exploited in most applications due to the relative underdevelopment of compilers, which despite many improvements still lag far behind. Due to this gap between compiler and processor technology, applications cannot exploit successfully *MCP* hardware unless they are explicitly aware of thread interactions. In this paper, we have revised the implementation of multigrid smoothers in this light. The popularity of multigrid makes this study of great practical interest. In addition, it also provides certain insights about the potential benefits of this relatively new capability and how to take advantage of it, which could ideally helps to develop more efficient compiler schemes.

The organization of this paper is as follows. We begin in Sections 2 by briefly introducing multigrid methods, in Section 3 summarizes the most relevant optimizations in Multigrid Algorithm in our context and Section 4 describes the main characteristics of our target computing platform. Afterwards, in Section 5, we discuss our *MCP*-aware implementation. Performance results are discussed in Section 6. Finally, the paper ends with some conclusions.

2 Multigrid Introduction

The fundamental idea behind Multigrid methods[1] is to capture errors by utilizing multiple length scales (multiple grids). They consist of the following complementary components:

- *Relaxation*. Also called smoother in multigrid lingo, is basically a simple (and inexpensive) iterative method like *Gauß-Seidel*, damped *Jacobi* or block *Jacobi*. It is able to reduce the high-frequency or oscillatory components of the error in relatively few steps.
- *Coarse-Grid Correction*. Smoothers are ineffectual in attenuating low-frequency content of the error, but since the error after relaxation should lack the oscillatory components, it can be well-approximated using a coarser grid. On that grid, errors appear more oscillatory and thus the smoother can be applied effectively. New values are transferred afterwards to the target grid to update the solution.

The *Coarse-Grid Correction* can be applied recursively in different ways, constructing different cycling strategies. One of the most employed cycles correspond to the V-cycle.

3 Related Work

The optimization of multigrid codes has been a popular research topic over the last years. For instance, one of the most outstanding and systematic studies is the *DIME* project (*DIME* stands for *Data Local Iterative Methods For The Efficient Solution of Partial Differential Equations*) [11,7–10]. Most optimization has been focused on the smoother, which is typically the most time consuming part of a multigrid method, and specifically on the red-black Gauß-Seidel method, which is by far one of the most popular smoothers. Our study is based on this previous research. In fact, as baseline codes we have employed the highly optimized variants of a two-dimensional red-black Gauß-Seidel relaxation algorithm developed within the *DIME* project [11]. This naïve implementation performs a complete sweep through the grid for updating all the red nodes, and then another complete sweep for updating all the black nodes. Therefore, *rb1* exhibits lower spatial locality than a lexicographic ordering. Furthermore, if the target grid is large enough, temporal locality is not fully exploited.

Alternatively, some authors have successfully improved cache reuse (locality) using loop reordering and data layout transformations that were able to improve both temporal and spatial data locality[12,7].

Following these previous studies, in this paper we have used as baseline codes the different red-black smoothers developed within the framework of the *DIME* project. To simplify matters, these codes are restricted to 5-point as well as 9-point discretization of the Laplacian operator. Figures 2-4 illustrate some of them which tries to fuse the *red* and *black* sweeps.

On the other hand, to the author's knowledge, the study of multigrid smoothers on Multicore Architectures has been hardly researched previously. In[13], authors addressed the alternative parallelization of a 3D-Multigrid Smoother in a *MCP* architectures which is based on temporal blocking scheme. The main idea consists in a temporal division of each slice (group of points) into blocks, which are assigned in the proper order to the

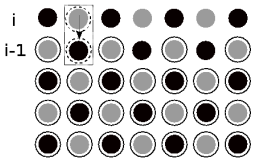

Figure 2. *DIME's* rb2. The update of red and black nodes is fused to improve temporal locality.

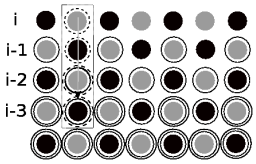

Figure 3. *DIME's* rb5. Data within a tile is reused as much as possible before moving to the next tile.

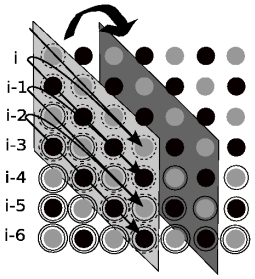

Figure 4. *DIME's* rb9. Data within a tile is reused as much as possible before moving to the next tile.

— — — —

Table 1. Main features of the target computing platform.

Processor	Intel Core^{TM}2 Duo 2.4 GHz	
	L1 DataCache	32+32 KB (data+instruction) 8-way set associative
	L2 Unified Cache	4 MB 8-way set associative
Memory	2048 MBytes DDR2-533 MHz SDRAM	
Operating System	GNU Debian Linux kernel 2.6.20-SMP for 32 bits	
Intel Fortran and C/C++ Compiler Switches(v9.1)	-static -O3 -tpp7 -xT -ip -ipo -no-prec-div Parallelization with OpenMP: -qsmp=omp	

different threads as it was a pipeline approach. Each block is marked with the numbers of relaxations which allows the multiple relaxations in each sweep. However, despite the relative simplicity of the proposed scheme, it is not compatible with some of the most successfully sweeps developed in the *DIME* project. Therefore, it makes highly necessary the revision of the exploitation in a *MCP* architectures in conjunction with better memory usage exposed in the *DIME* project.

4 Experimental Platform

Our experimental platform consists in an *Intel Core^{TM}2 Duo* processor running under Linux, the main features of which are summarized in Table 1.

With this design, this dual-core includes two independent microprocessors that share some resources such as L2 memory cache and the main memory access.

Finally, it is worth to mention that the exploitation of *MCP* has been performed in this work by means of *OpenMP* directives, which are directly supported by the Intel FORTRAN/C native compilers[14].

5 MCP-aware Red-Black Smoothers

The availability of *MCP* introduces a new scenario in which thread-level parallelism can be exploited by means of the execution of several threads in the different cores. This logical view suggests the application of the general principles of data partitioning to get the multithreaded versions of the different *DIME* variants of the red-black Gauß-Seidel smoother. This strategy, which can be easily expressed with *OpenMP* directives, is suitable for shared memory multiprocessor. However, in a *MCP*, the similarities amongst the different threads (they execute the same code with just a different input dataset) may cause contention since they have to compete for the shared L2-cache and memory accesses.

Algorithm 4 Interleaved implementation of a red-black Gauß-Seidel

```
#pragma omp parallel private(task,more_tasks) shared(control_variables)
more_tasks = true
while more_tasks do

  #pragma omp critical
  Scheduler.next_task(&task);

  if (task.type == RED) then
    Relax_RED_line(task);
  end if

  if (task.type == BLACK) then
    Relax_BLACK_line(task);
  end if

  #pragma omp critical
  more_task=Scheduler.commit(task);

end while
```

Alternatively, we have employed a dynamic partitioning where computations are broken down into different tasks with are assigned to the pool of available threads. Intuitively, the smoothing of the different colours is interleaved by assigning the relaxation of each colour to a different thread. This interleaving is controlled by a scheduler, which avoids race conditions and guarantees a deterministic ordering.

Algorithm 4 shows a pseudo-code of this approach for red-black smoothing. Our actual implementation is based on the *OpenMP's parallel* and *critical* directives. The critical sections introduce some overhead but are necessary to avoid race-conditions. However, the interleaving prompted by the scheduling allows the *black thread* to take advantage of some sort of data prefetching since it processes grid nodes that have just been processed by the *red thread*, i.e. the *red thread* acts as a helper thread that performs data prefetching for the *black* one.

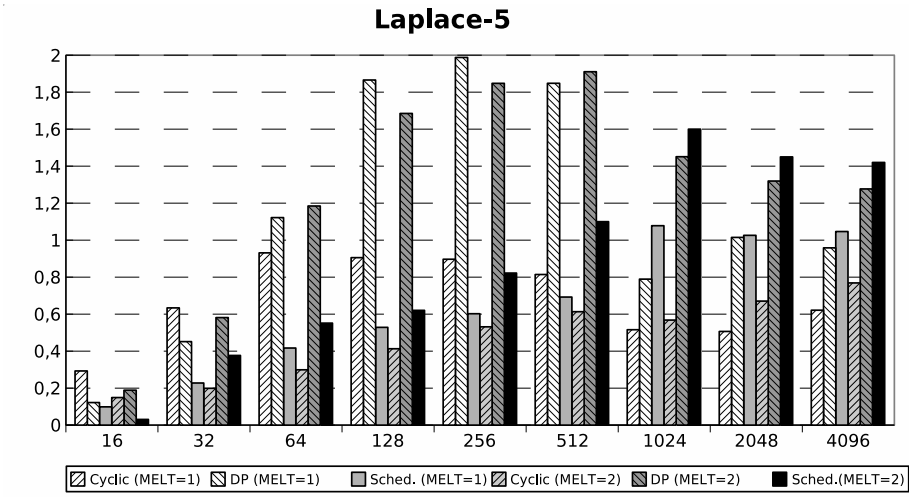

Figure 5. Speedup achieved by different parallel implementations of a red-black Gauß-Seidel smoother for a 5-point Laplace stencil. Sched denotes our strategy, whereas DP and Cyclic denote the best block and a cyclic distribution of the smoother's outer loop respectively. MELT is the number of successive relaxations that have been applied.

6 Results

Figure 5 shows the speedup achieved by the different parallel strategies over the baseline code (with the best DIME's transformation) for the the 5-point stencil. Our strategy improves inter-thread locality taking advantage of fine-grain thread sharing especially in large grid sizes.

As can be noticed, the election of the most suitable strategy depends on the grid size:

- For small and medium grid sizes block and cyclic distributions outperforms our approach, although for the smallest sizes none of them is able to improve performance as consequence of the costs involved in the creation/destruction of thread which is not compensated by the parallel execution. For these working sets, memory bandwidth and data cache exploitation are not a key issue and traditional strategies beats our approach on performance due to the overheads introduced by the dynamic task scheduling.

- However, for large sizes we observe the opposite behaviour given that the overheads involved in task scheduling become negligible, whereas the competition for memory resources becomes a bottleneck in the other versions. In fact, we should highlight that the block and cyclic distributions become less efficient for large grids.

- The break-even point between the static distributions and our interleaved approach is a relative large grid and corresponds to 4 MB (L2 shared cache of the *Intel Core^{TM}2 Duo*).

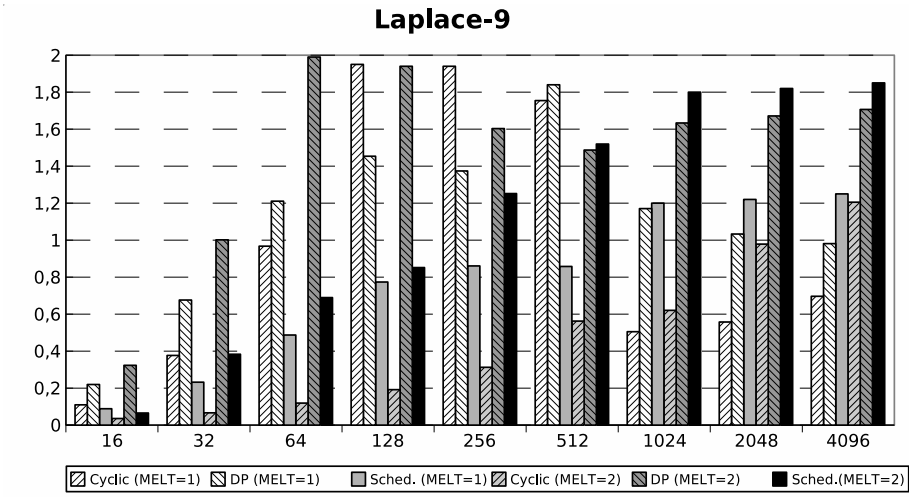

Figure 6. Speedup achieved by different parallel implementations of a red-black Gauß-Seidel smoother for a 9-point Laplace stencil. Sched denotes our strategy, whereas DP and Cyclic denote the best block and a cyclic distribution of the smoother's outer loop respectively. MELT is the number of successive relaxations that have been applied.

Figure 6 confirms some of these observations for the the 9-point stencil. Furthermore, the improvements over *DIME's* variants are higher in this case, since this is a more demanding problem.

7 Conclusions

In this paper, we have introduced a new implementation of red-black Gauß-Seidel Smoothers, which on *MCP* processors fits better than other traditional strategies. From the results presented above, we can draw the following conclusions:

- Our alternative strategy, which implicitly introduce some sort of tiling amongst threads, provide noticeable speedups that match or even outperform the results obtained with the different *DIME's rb2-9* variants for large grid sizes. Notice that instead of improving *intra-thread locality*, our strategy improves locality taking advantage of fine-grain thread sharing, expecially successful in the caches shared as in most of *MCPs*.

- For large grid sizes, competition amongst threads for memory bandwidth and data cache works against traditional block distributions. Our interleaved approach performs better in this case, but suffers important penalties for small grids, since its scheduling overheads does not compensate its better exploitation of the temporal locality. Given that multigrid solvers process multiple scales, we advocate hybrid approaches.

Acknowledgements

This work has been supported by the Spanish research grants TIC 2002-750, TIN 2005-5619 and the Hipeac European Network of Excellence.

References

1. U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, (Academic Press, 2000).
2. J. L. Thomas, B. Diskin and Achi Brandt, *Textbook multigrid efficiency for fluid simulations*, Annual Review of Fluid Mechanics, **35**, 317–340, (2003).
3. M. F. Adams, M. Brezina, J. J. Hu, and R. S. Tuminaro, *Parallel multigrid smoothing: polynomial versus Gauss-Seidel*, J. Comp. Phys., **188**, 593–610, (2003).
4. E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro and U. M. Yang, *A Survey of parallelization techniques for multigrid solvers*, Tech. Rep., (2004).
5. W. Mitchell, *Parallel adaptive multilevel methods with full domain partitions*, App. Num. Anal. and Comp. Math, **1**, 36–48, (2004).
6. F. Hülsemann, M. Kowarschik, M. Mohr and U. Rüde, *Parallel Geometric Multigrid*, Lecture Notes in Computer Science and Engineering, **51**, 165–208, (2005).
7. C. Weiß, W. Karl, M. Kowarschik and U. Rüde, *Memory characteristics of iterative methods*, in: Proc. ACM/IEEE Supercomputing Conf. (SC99), Portland, Oregon, USA, (1999).
8. M. Kowarschik, U. Rüde, C. Weißand W. Karl, *Cache-aware multigrid methods for solving Poisson's equation in two dimensions*, Computing, **64**, 381–399, (2000).
9. C.C. Douglas, J. Hu, M. Kowarschik, U. Rüde and C. Weiß, *Cache optimization for structured and unstructured grid multigrid*, Electronic Transactions on Numerical Analysis (ETNA), **10**, 21–40, (2000).
10. M. Kowarschik, C. Weißand U. Rüde, *Data layout optimizations for variable coefficient multigrid*, in: Proc. 2002 Int. Conf. on Computational Science (ICCS 2002), Part III, P. Sloot, C. Tan, J. Dongarra, and A. Hoekstra, (Eds.), of *Lecture Notes in Computer Science*, vol. **2331**, pp. 642–651, (Springer, Amsterdam, 2002).
11. Friedrich-Alexander University Erlangen-Nuremberg. Department of Computer Science 10, DIME project, Available at http://www10.informatik.uni-erlangen.de/Research/Projects/DiME-new.
12. D. Quinlan, F. Bassetti and D. Keyes, *Temporal locality optimizations for stencil operations within parallel object-oriented scientific frameworks on cache-based architectures*, in: Proc. PDCS'98 Conference, (1998).
13. D. Wallin, H. Löf, E. Hagersten and S. Holmgren, *Multigrid and Gauss-Seidel smoothers revisited: parallelization on chip multiprocessors*, in: ICS '06: Proc. 20th Annual International Conference on Supercomputing, pp. 145–155, (ACM Press, New York, 2006).
14. Intel Corpation, em Intel C/C++ and Intel Fortran Compilers for Linux, Available at http://www.intel.com/software/products/compilers.